ERDC/CRREL TR-09-17

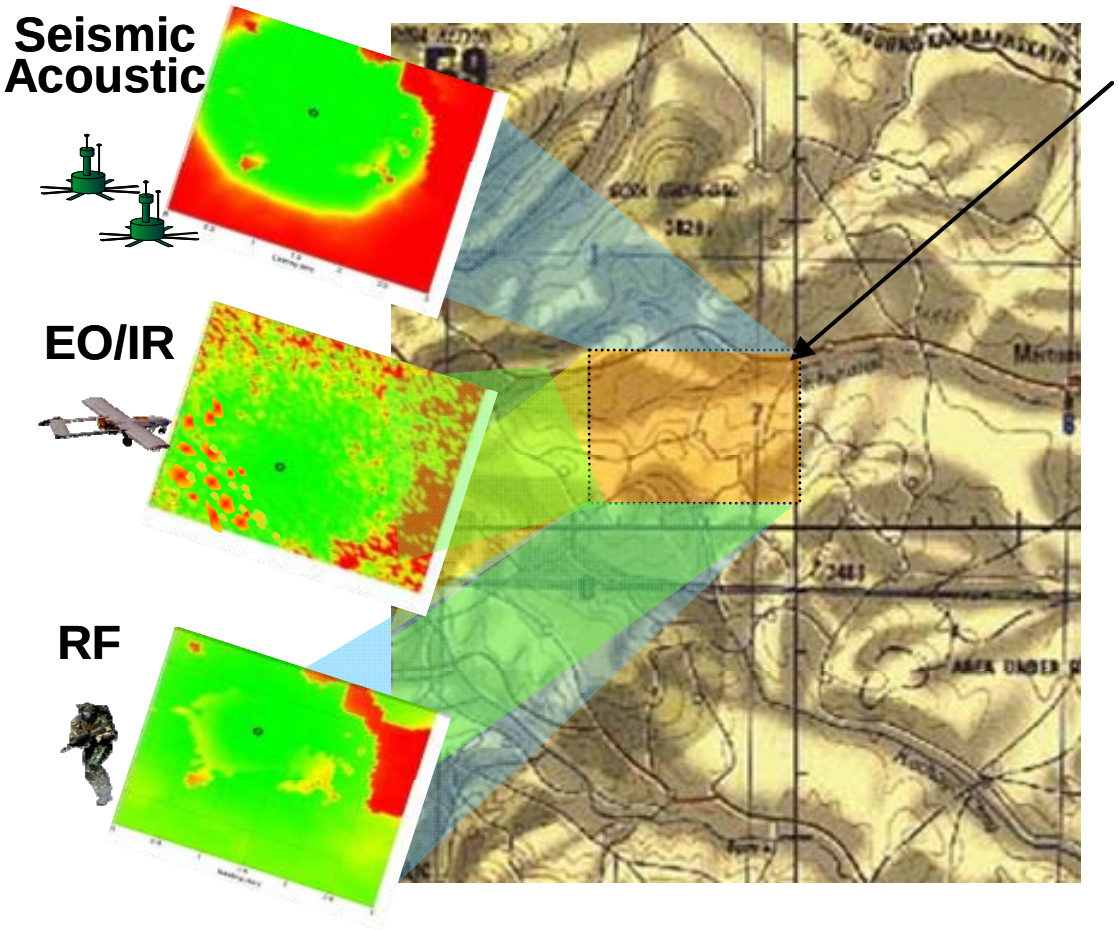
Cold Regions Research and Engineering Laboratory

US Army Corps of Engineers®
Engineer Research and Development Center

Object-Oriented Software Model for Battlefield Signal Transmission and Sensing

D. Keith Wilson, Richard Bates, and Kenneth K. Yamamoto

December 2009



Geospatial Research and Engineering

ERDC/CRREL TR-09-17
December 2009

An Object-Oriented Software Design for Battlefield Signal Transmission and Sensing

D. Keith Wilson, Richard Bates, and Kenneth K. Yamamoto

Cold Regions Research and Engineering Laboratory
U.S. Army Engineer Research and Development Center
72 Lyme Road
Hanover, NH 03755-1290

Final report

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

Under AT42 GEOINT Exploitation in Man-Made Environments: Nations to Insurgents (GEMENI)

Abstract: Realistic modeling and simulation of battlefield signal transmission and sensing requires accounting for many complicated environmental and mission-related factors. These factors present complex challenges because of their diversity. Yet, successful modeling and simulation can enable effective mission planning and can support a variety of other military objectives. This report describes the development of a very flexible, object-oriented software design for predicting signal transmission and sensing on the battlefield. This Java-language software is called Environmental Awareness for Sensor and Emitter Employment (EASEE). It is intended for application to a wide range of sensing modalities, and for incorporation into military command and control (C2) systems, decision support tools (DSTs), and force-on-force simulations. An initial version of a user interface for EASEE has been completed and a Java OpenMap implementation has begun. The goal of this work is to make EASEE general enough to be implemented within many software architectures, which will enable advanced signal propagation and processing calculations in a number of modeling and simulation efforts in support of the warfighter and homeland security.

Contents

Figures and Tables iv

Preface v

1 Introduction 1

Background 1
Objectives 2
Tech transfer 2
Outline of information 2

2 Object-Oriented Software Design 4

Introduction to object-oriented programming 4
An object-oriented view to signal transmission and sensing 6

3 Signature Data Features 8

Signal model objects 8
Feature definitions 11

4 Inference Objects 13

5 From Feature Generation to Inference Fusion 15

Information flow 15
Software implementation 17

6 Platforms: Sensor and Emitter Representation 20

Configuration of a platform 20
Software implementation 21
Platform positioning and problem viability 23

7 Environmental Representation 26

8 Signal Transmission Grids 28

9 Conclusions 31

References 32

Appendix: Overview of EASEE Java Packages 34

Report Documentation Page

Figures and Tables

Figures

Figure 1. Illustration of inheritance and polymorphism. Orange and apple are subclasses of fruit. Macintosh and Gala are subclasses of apple. Left: Methods written for fruit accept all classes denoted in green, which includes fruit, orange, apple, Macintosh, and Gala. Right: Methods written for apple accept apple, Macintosh, and Gala. 6

Figure 2. Statistical signal model inheritance tree. 11

Figure 3. Inference model inheritance tree. Black boxes indicate inferences that are formed from signal models; green boxes indicate inferences that are formed (fused) from other inferences. 14

Figure 4. Flow diagram for processing of signature data features and inferences. The signature data features are *generated*, *propagated* through the environment, *sensed*, and then *processed* into inferences. The inferences may then be *fused* into other inferences. Solid boxes indicate functionality of *abstract objects*, whereas the dashed lines are *interfaces*. 15

Figure 5. Configurations of feature transmitters and processors to represent a wheeled ground vehicle (left) and an unattended aerial vehicle (right). 21

Figure 6. Configurations of feature and fusion processors to represent an intrusion detection sensor (left) and a ground-sensor gateway/fusion node (right). 21

Figure 7. Derivation of the *GammaSignalModel* class, showing its parent classes and inherited methods. 34

Figure 8. Derivation of the detection feature and fusion processors, showing parent classes and inherited methods 35

Preface

This study was conducted for the U.S. Army Corps of Engineers. Funding was provided by the Engineer Research and Development Center (ERDC) Geospatial Intelligence (GEOINT) program, Exploitation in Man-made Environments: Nations to Insurgents (GEMENI).

The work was performed by the Signature Physics Branch (CEERD-RR-D) of the Research and Engineering Division (RR) at the U.S. Army Engineer Research and Development Center – Cold Regions Engineering Research Laboratory (ERDC-CRREL). The principal investigator was Dr. D. Keith Wilson (RR-D). The graphic on the title page of this report was created by Dr. Dale R. Hill (ERDC-CRREL). The authors thank Dr. George G. Koenig of ERDC-CRREL's Terrestrial and Cryospheric Sciences Branch (CEERD-RR-G) for providing helpful comments on a draft version of this report.

At the time of publication, Dr. John M. Boteler was Chief, CEERD-RR-D; Dr. Justin B. Berman was Chief, CEERD-RR; and Dr. Dale R. Hill was the Acting Technical Director for Geospatial Research and Engineering. The Deputy Director of ERDC-CRREL was Dr. Lance D. Hansen, and the Director was Dr. Robert E. Davis.

COL Gary E. Johnston was the Commander and Executive Director of ERDC, and Dr. James R. Houston was the Director.

1 Introduction

Background

The performance and utility of battlefield and homeland security sensors depends on many complex factors, both environmental and mission-related. This is generally true whether the sensors are ground-based or airborne; whether they are acoustic, seismic, optical, infrared (IR), radio frequency (RF), or magnetic; and whether the signal emitters originate from vehicles, humans, or electronic equipment. Realistic modeling and simulation of factors that influence sensing and signal emission can enable effective mission planning, improve virtual prototyping of sensor systems and signal processing algorithms, and support force-on-force simulations and doctrinal development.

An additional challenge to develop advanced modeling and simulation capabilities is the diversity of sensors and signal emitters on the battlefield (including targets of interest plus natural and man-made interferences), and the environmental phenomena affecting those sensors and emitters.

As an example, let us contrast modeling of acoustic and IR sensors. The following paragraph illustrates why different sensing modalities are often modeled by very different approaches, and those different approaches may be challenging to reconcile.

Acoustic signatures produced by vehicles typically depend strongly on the vehicle type and its operation. Environmental variables such as temperature and solar loading have a minor effects on the acoustic signature. However, the impact of weather and terrain on acoustic signal propagation is very significant, because wind and temperature gradients refract the sound waves, and the sound diffracts around hills and buildings. Microphones usually have a nearly omnidirectional response that does not vary substantially with environmental conditions. On the other hand, passive IR signatures depend on differential heating and cooling of the target and background, which is a function of solar irradiation, evaporation, wind flow, and other factors. The atmospheric effect on the signal transmission (the optical depth) is often a secondary concern, at least for short transmission paths in clear weather. For an IR sensor, the directionality (field of view) is the important consideration.

Objectives

The purpose of this report is to describe a software design for predicting the performance of various types of battlefield sensors and detecting various types of emitters. We developed a design that is general enough to readily incorporate the diverse factors illustrated in the preceding paragraph. Although our emphasis is on decision support tools (DSTs) for battlefield command and control (C2) applications, and incorporation of such tools into geospatial information systems (GIS) (Wilson et al. 2007; Hieb et al. 2007; Frankenstein and Koenig 2004), our broader intention is to produce a highly flexible software design that can be used in a great variety of modeling and simulation applications.

Tech transfer

EASEE is intended for application to a wide range of sensing modalities, and for incorporation into military command and control (C2) systems, decision support tools (DSTs), and force-on-force simulations. An initial version of a user interface for Environmental Awareness for Sensor and Emitter Employment (EASEE) has been completed and a Java OpenMap implementation has begun. The goal of this work is to make EASEE general enough to be implemented within many software architectures, which will enable advanced signal propagation and processing calculations in a number of modeling and simulation efforts in support of the warfighter and homeland security.

Outline of information

The software design is called Environmental Awareness for Sensor and Emitter Employment, or EASEE. This report describes the structure of EASEE. We begin, in Section 2, with a short introduction to object-oriented programming (OOP), which underlies the EASEE design. This section also describes the key components of DSTs such as EASEE. This discussion motivates the central role of statistical models for signature data features, which is the topic of Section 3, and statistical models for inferences, which is the topic of Section 4. Section 5 discusses the information flow from generation of the signature data features to fusion of inferences, and how this is implemented in EASEE. Section 6 describes how the various modeling components outlined in Section 5 are built into model emitter and sensor platforms. Representation of the environment (atmosphere and terrain) is the topic of Section 7. Efficient representation of sig-

nal transmission is discussed in Section 8. Finally, information on specific Java software packages comprising EASEE is provided in the Appendix.

2 Object-Oriented Software Design

Introduction to object-oriented programming

The EASEE software design is formulated within the conceptual framework of object-oriented programming (OOP). In this section, we provide a brief introduction to OOP, particularly as it is implemented in the Java programming language. (Readers unfamiliar with OOP or Java may find this helpful in understanding the remainder of the report.)

OOP is a conceptual approach to programming. Many programming languages introduced over the past 20 years or so, including Java, C++, and C#, were specifically designed to facilitate OOP. OOP techniques can be implemented in other languages, but often the coding is more cumbersome and the result less elegant than for a dedicated OOP language.

OOP can perhaps be best understood by contrasting it to more familiar *modular* programming in languages such as FORTRAN 77, Pascal, and C. Good programming practice in these languages generally involves partitioning a more complicated task into a number of subtasks, with each subtask being implemented in a function, procedure, or subroutine. Hence the emphasis is on the algorithmic formulation rather than the data upon which the algorithms operate. OOP, in contrast, endeavors to group data into collections that represent particular entities, or *objects*. Tasks, or *methods,* are then associated with these objects. This association between data collections and the tasks performed upon them is the main idea behind OOP.

Objects possess the following characteristics (Lowe 2005):

- *Type*. Some examples of object types are a person, an automobile, and a geographic coordinate. In Java, types are called *classes*.
- *Identity*. With regard to the types mentioned above, corresponding example identities would be Bob, my Chevrolet, and the location of CRREL. In Java, objects of a particular type are said to be *instances* of the class.
- *State*. Corresponding to the above, these might be cheerful, parked, and 43.697 N 72.311 W. In Java, the *fields* (variables) of the class describe its state.

- *Behavior*. Corresponding to the above, these might be hard working, reliable, and coordinate conversions. In Java, *methods* describe the behavior of the class.

Some other important concepts related to OOP are *inheritance*, *instantiation*, *abstraction*, *overloading*, *visibility*, and *polymorphism*.

Inheritance refers to the ability to use one class as a basis for designing another. For example, a class representing fruit may be the basis for one representing apples, which may in turn be the basis for one representing McIntosh applies (Figure 1). The "apple" class is said to be a *subclass* (or extension or child) of the *superclass* (or base or parent) "fruit." When a subclass redefines a method (behavior) defined in the parent class, that method is said to be *overloaded*.

When an object of a class is created, it is said to be *instantiated*. In Java, *constructor* methods are used to create objects and initialize their fields if desired. An abstract class is one that cannot be instantiated. However, it can still serve as the basis for subclasses that can be instantiated.

The *visibility* of a field or method refers to its availability outside of the class in which it is defined. In Java, the keyword `public` identifies a field or method that can be accessed outside of the class. Fields and methods that are `protected` can be accessed only within the class or by subclasses, whereas `private` fields and methods can be accessed only within the class.

Polymorphism refers to the ability of objects of a subclass to take on the identity of objects of the parent class. This is actually a key reason why object-oriented languages can be such powerful tools. When code is written for the highest-level object possible, all lower-level objects can automatically utilize this code. This approach is consistent with scientific principles of identifying and modeling general features and behaviors. It is illustrated in Figure 1. In Java, the classes of objects can be explicitly changed if desired; this is called *casting*.

How does this discussion of object-oriented programming relate to our goal of developing a general software design accommodating different signal types, propagation models, and weather and terrain inputs? Essentially, the idea is to *abstract the functionality to the highest level possible*

through application of OOP concepts of *inheritance* and *polymorphism*. As will become clearer in the following discussion, such an approach greatly facilitates the development of general purpose code.

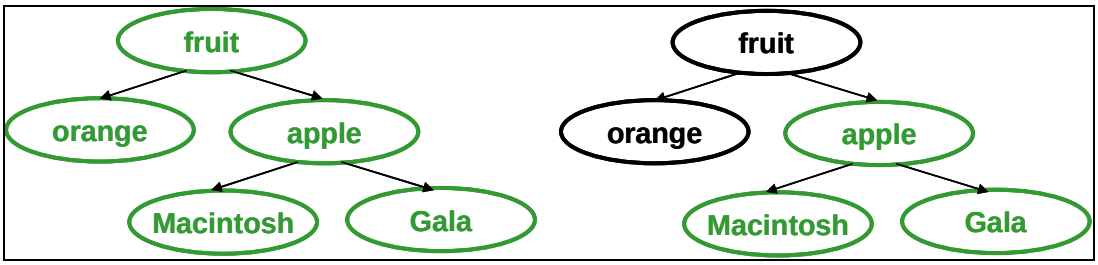


Figure 1. Illustration of inheritance and polymorphism. Orange and apple are subclasses of fruit. Macintosh and Gala are subclasses of apple. Left: Methods written for fruit accept all classes denoted in green, which includes fruit, orange, apple, Macintosh, and Gala. Right: Methods written for apple accept apple, Macintosh, and Gala.

An object-oriented view to signal transmission and sensing

Wilson et al. (2008) suggested that a DST for predicting battlefield sensor performance (or the reciprocal problem, detectability of battlefield targets) involves the following steps: (1) *information gathering* and construction of the environmental and tactical scenario, (2) *translation* of this information into parameters needed by signal and noise prediction models, (3) *signal and noise prediction* (providing a parametric description of the signal and noise from the emitter characteristics, signal propagation, and sensor processing), (4) calculation of sensor *performance metrics* (e.g., probability of detection or accuracy of target localization), and (5) *display of* and *interaction with* the information (graphical interface).

Although all of these steps are important in the design of a satisfactory DST, steps (3) and (4) are the primary steps targeted by the EASEE architecture. They involve prediction and interpretation, respectively, of the sensor-related information. The essence of these steps is that a sensor receives a "raw" data stream containing the combined signal from the emitter of interest as well as interfering background noise (clutter). This data stream is then processed into a set of *signature data features* which are used to infer characteristics of the signal emitter(s) of interest. Hence prediction of these sensed features (step 3) and the inferences made from them (step 4) can be considered the key parts of the problem. This discussion suggests that an object-oriented approach to battlefield signal transmission and processing involves two fundamental types of objects: *signature data features* and *inferences*.

As will be seen, the EASEE architecture is also designed to facilitate step (2), translation of the environmental information, as needed to support steps (3) and (4). However, EASEE itself does not perform steps (1) and (5). We envision that these will be performed by a separate software program that interfaces with EASEE, which we call here the *user application interface*. EASEE is designed to be transportable between many such interfaces.

The next question to be considered is what characteristics of the signature data features are actually needed to predict sensor performance? In general, we cannot predict the precise values of the features in a particular circumstance, as the signals are subject to random generation mechanisms and propagation effects. Thus we model and predict their statistical distributions; the inferences should then also be viewed in a probabilistic sense.

The identification of signature data features and inferences as the primary items of interest may, in retrospect, seem obvious. However, DSTs have not normally been built explicitly upon this simple premise. On the contrary, models for signal propagation physics, noise backgrounds, and signal processing involve a wide variety of implicit and explicit statistical assumptions that tend to vary greatly with the particular field (acoustics, optics, radio frequency, etc.). Simulations are often built around generation and processing of raw signals, a practice that can be very computationally expensive for C2 applications. When statistical models are used, different ones may be appropriate for describing different types of target signatures, propagation effects, and signal processing formulations. The following two sections provide more detail on the concepts of signature data feature and inference objects, as implemented in EASEE.

3 Signature Data Features

Signature data feature is a widely used term, but rather difficult to define precisely. The term generally refers to information extracted from a signal that might identify the source of that signal. Signature data features are *not* ordinarily the raw sensor data, but rather result from some low-level processing of raw data, such as calibrations to remove the sensor response and filtering of the signal into spectral bands. Other examples might include the sound power of a harmonic line or in a standard octave band; IR brightness in near, shortwave, midwave, longwave, or far bands; components of an electromagnetic field vector; or concentration of a chemical or biological species. Typically, the feature would represent a *conservative* quantity, such as energy or mass. In the literature on sensor data processing, signature data features as described here can be associated with the output of level-zero information processing (Klein 2004).

Signal model objects

In general, signature data features vary randomly. The generation process is often irregular, and propagation through the environment is subject to random effects such as turbulence. Thus it is necessary to describe the features with statistical models. Some examples of appropriate statistical models include Gaussian (normal), exponential, gamma, lognormal, Rayleigh, Rice-Nakagami, central chi-square, and noncentral chi-square probability density functions (pdfs). (Examples are in Burdic 1984; Wilson et al. 2002; and Nadarajah 2008.)

The *Gaussian* model, while advantageous for its simplicity and tractability, can reasonably be applied to signal power only when the standard deviation is much less than the mean; otherwise, the signal power would frequently be negative, which is not physical.

For randomly scattered signals (e.g., acoustic or RF scattering by buildings, vegetation, and turbulence), other pdfs are appropriate. The *exponential* pdf describes a single, strongly scattered signal, and is a special case of the *central chi-square* pdf, which describes the sum of multiple, strongly scattered signals. The *noncentral chi-square* pdf can include a deterministic (non-zero mean) contribution. Since the *lognormal* pdf models variables that can be the multiplicative product of many independent,

positive random variables, it is a good approximation of many atmospheric physical and chemical properties, including the plume size and frequency distributions of transient gases from turbulent processes (Baker et al. 1983; Limpert et al. 2001). Hence, the *lognormal* pdf is well suited to representing concentrations of chemical or biological species that may be dispersed into the atmosphere.

In some cases, the features at a sensor may not be statistically independent. The pdf should then actually be a joint pdf between multiple kinds of sensed data. In principle, the multiple sensed data features could be regarded and manipulated by processing algorithms as a single entity. To implement predictions involving a sensor array, such as bearing estimation, joint signal models including spatial correlation functions of the signals would be necessary.

Stitching together disparate statistical formulations, like those described in the preceding paragraphs, is a challenging programming task. But, it must be done if very general software designs are to be successfully developed. A parametric description of the pdf of a signature feature may be regarded as a programming object, the propagation and manipulation of which is central to EASEE.[1] In Java, it is natural to represent each distribution as a class, with each class defining the model parameters for the signature features. We call these *signal-model classes*. The signal-model class specifies how the parametric pdfs are to be manipulated; in particular, a method for calculating the pdf itself must be supplied. Optionally, translators to explicitly convert parametric pdfs from other classes may be included, although when possible the conversion is made automatic when signal-model objects are cast from one class to another.

Classes can implement certain manipulations of the pdfs that are useful for probability of detection and other types of information processing. For example, most probability of detection calculations involve integrating the pdf for signal power between zero and a threshold value (the detector threshold), or between the threshold value and infinity. (In statistics, these are called cumulative distribution functions, or cdfs.) Determination of the threshold (quantile, or inverse cdf) corresponding to a specified probability is also of interest. These operations are logically associated with the

[1] Since probability density functions, cumulative density functions, quantiles, and other statistical concepts are discussed in detail elsewhere, we do not introduce these concepts here. Wilks (2005) is highly recommended to readers interested in an introduction to this subject.

signal-model class, rather than with sensor feature processors. In this manner they can be reused by multiple processors.

The signal-model class must also specify how to calculate the pdf of the combination (sum) of two signals. This is essential to the overall architecture, since multiple signals of interest and interfering signals may arrive at a sensor. The calculation is trivial for Gaussian (normally distributed) signals, in which case the mean and variance of the sum signal are found by summing the means and variances of the contributing signals. Although the Gaussian model is simple and tractable, it is unrealistic for many situations, as discussed earlier. Unfortunately, for other known models commonly used to describe random signals, such as the exponential, chi-square, and Rice-Nakagami distributions, the pdf of the sum has a different functional form than the pdf of the contributing signals, and in some cases, no analytical result is available at all. In many situations there appears to be no alternative to approximating the pdf of the combined signal.

For efficiency, all methods in the signal-model classes are parallelized to operate on multiple data points, which would normally represent different locations or times. That is, each parameter in the pdf is given by an array of values, each element of which corresponds to the pdf parameter for a different location or time.

Inheritance plays an important role in the EASEE signal-model implementation. A base signal-model class, called `AbstractSignalModel`, is defined with abstract methods for calculating signal means, variances, pdfs, cdfs, and quantiles. Extensions (subclasses) of the abstract base class implement these operations. By convention, these subclasses are assigned names like `GaussianSignalModel`, in which case a Gaussian pdf is used to describe the signal statistics. Similarly, `GammaSignalModel` would use a gamma-function pdf. This structuring of the signal-model classes, in which the actual statistical models are extensions of a parent abstract class, is central to the entire EASEE architecture. It allows sensor data processing models to be written in a way that is general for any statistical model for the signals.

An object tree illustrating a set of signal model classes and inheritance relationships is shown in Figure 2. The abstract class for constant (time-invariant) signal models extends the abstract base class (`AbstractSignalModel`) directly. The constant signal model class

(called `ConstantSignalModel`) involves just one parameter, namely the signal mean. Four other signal-model classes extend this class to variable signals possessing other pdf parameters in addition to the mean: these are for discrete-valued (multiple, fixed values), exponential, Gaussian, and log-normal pdfs. The gamma signal model extends the exponential.

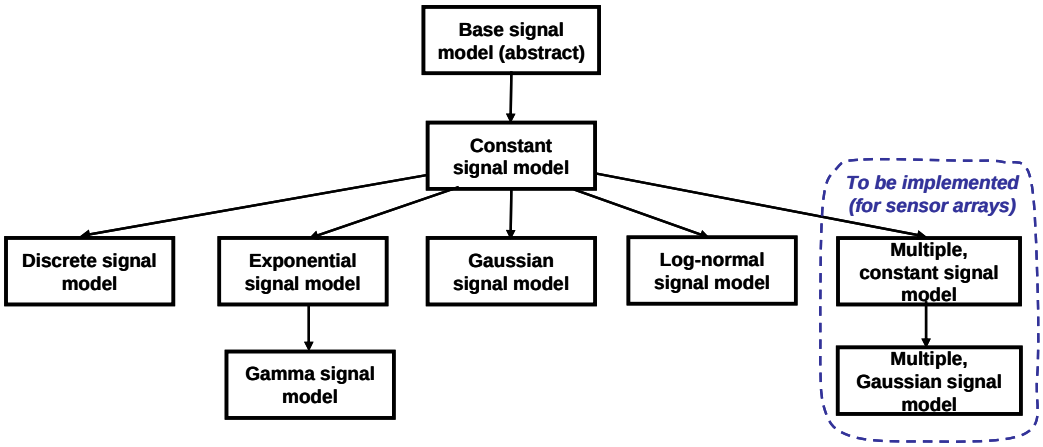


Figure 2. Statistical signal model inheritance tree.

The signal model classes like `GaussianSignalModel` and `GammaSignalModel` are non- abstract – they implement all of the functionality prescribed by the `AbstractSignalModel`. However, they do not define the signature data features to which they apply.

Feature definitions

In EASEE, the definition of signature data features is independent from the signal model classes used to represent them. The feature definition is specific to the kind of signal. Features are defined through Java enumerations (`enum` types), which list the applicable signature features and their properties.[2] This design makes it possible for multiple kinds of signals to use a common signal model. Enumerations are presently available for acoustic, optical, radio-frequency, seismic, chemical, and biological features. The acoustic and optical enumerations are described here to illustrate the general design.

[2] A Java `enum` type is an alphanumeric listing of the members of a set. For example, the suits in a deck of cards might be defined with a Java `enum` containing the names “spades,” “hearts,” “clubs,” and “diamonds.”

At present, two enumerations have been developed for acoustical features. `AcousticOctaveBandFeatures` defines standard acoustic octave and third-octave bands. The definition consists of a descriptive name, and the lower and upper frequency bounds for the band. For example, the `ACOUSTIC_OCTAVE_63HZ` has frequency limits from $63/(2^{1/3})$ to $63(2^{1/3})$, whereas the `ACOUSTIC_THIRD_OCTAVE_63HZ` has frequency limits from $63/(2^{1/6})$ to $63(2^{1/6})$. Methods are defined for calculating the arithmetic and geometric center frequencies. `AcousticSpectralBandFeatures` is similar, except that it defines 1 Hz (constant bandwidth) bands, from 0–500 Hz. This enumeration is useful for representing power spectra produced by Fourier transforms.

The optical enumerations are provided by `OpticalSignalFeatures`. Twenty-seven different optical bands are defined, which span the ultraviolet, visible, and infrared spectral regions. The optical bands are defined by their lower and upper wavelengths. For example, a visible feature `VISIBLE` is defined as extending from 380–750 nanometers (nm). For higher resolution applications, the visible region is broken down into six color regions, `VISIBLE_VIOLET, VISIBLE_BLUE, VISIBLE_GREEN, VISIBLE_YELLOW, VISIBLE_ORANGE,` and `VISIBLE_RED.` `VISIBLE_GREEN`, for example, extends from 495–570 nm. Additional spectral bands could be readily added to the Java enumerations; no other significant changes to the software would be needed to accommodate new feature definitions.

4 Inference Objects

As mentioned earlier, the signature data features are processed into inferences. Some examples of possible inferences are whether a target is present (a detection inference), what kind of target it is (a classification inference), and what direction the target is in (a bearing inference). Inferences may also be formed by fusing inferences from multiple sensors (e.g., seismic and IR detection inferences could be combined for a more robust detection inference). In an actual sensing system, the features would normally be processed directly to form a hypothesis or estimate, such as whether a target is present. For present purposes, however, we are concerned with probabilistic predictions of performance of sensors — e.g., *probability* of detection. The statistical description of an inference is specifically what we mean here by an *inference object*.

An inference object class has fields describing the inference statistics and related information, and methods for calculating the inference statistics. As with the signature data features, an abstract base class is defined at the root of the inheritance tree (Figure 3). This class, `AbstractInferenceModel`, includes methods for checking compatibility between inferences. It is extended by `PowerInferenceModel, DetectionInferenceModel`, and other inference classes.

`PowerInferenceModel` represents the mean and variance of the signal and noise powers at a sensor. These are to be inferred from the signal models at the sensor. Methods for retrieving and setting the means and variances, and for producing mean signal-to-noise ratios, are provided.

`DetectionInferenceModel` includes fields for the probabilities of detection and false alarm. Methods for producing the probabilities of false dismissal (missed detection) and correct dismissal are also provided, as are methods for combining detection inferences. The latter describe how to calculate the probabilities of detection and false alarm from multiple signature features (a feature processor) or from multiple inferences (a fusion processor).

For all inferences, a *null* value must be defined. The main idea is that when an inference is combined with a null inference, its value does not change. For power inferences, the value of the null inference is simply zero power. But, for other types of inferences, the null inference is not as obvious. For probability of detection inferences, for example, the null inference is defined as probabilities of detection and false alarm equal to zero.

Another important type of inference is the location of a source. Such inferences might be created by combining bearings and/or ranges from multiple sensor platforms. Support for this type of inference is not yet fully implemented in EASEE.

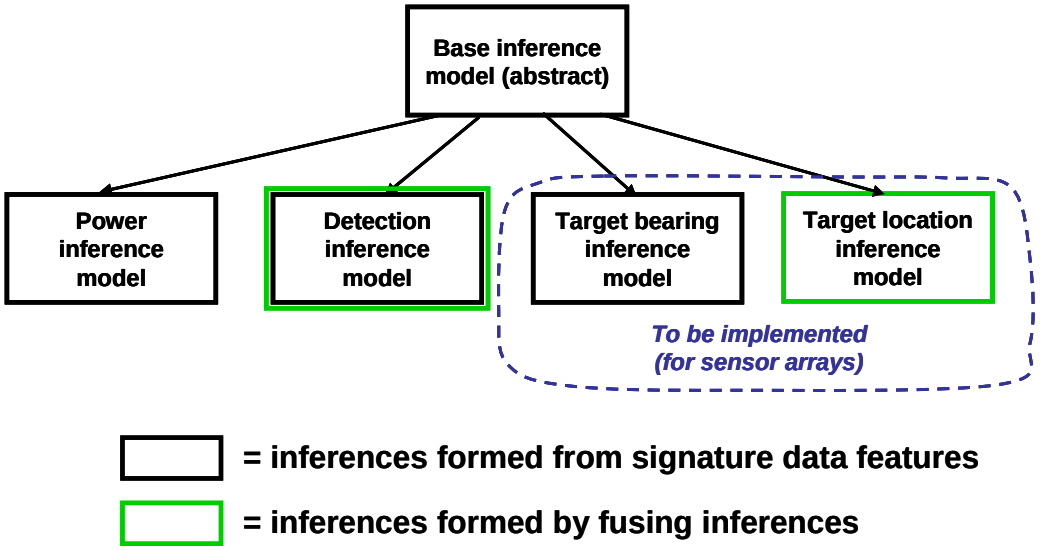


Figure 3. Inference model inheritance tree. Black boxes indicate inferences that are formed from signal models; green boxes indicate inferences that are formed (fused) from other inferences.

5 From Feature Generation to Inference Fusion

Information flow

As described up to this point, the focus of the EASEE software design is on statistics of signature data features (signal models), which represent units of information yielded by sensors. The signal models can be regarded as objects that must be *transmitted*, *received*, and *processed*, as illustrated in Figure 4.

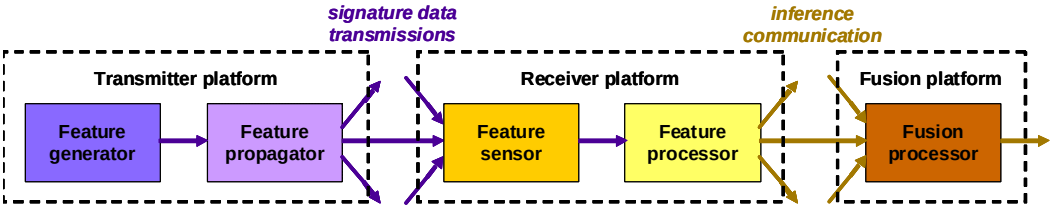


Figure 4. Flow diagram for processing of signature data features and inferences. The signature data features are *generated*, *propagated* through the environment, *sensed*, and then *processed* into inferences. The inferences may then be *fused* into other inferences. Solid boxes indicate functionality of *abstract objects*, whereas the dashed lines are *interfaces*.

A signature feature statistic generator, or *feature generator* for short, is a physical model for the production of signals observable by a sensor. Depending on the context, the feature generator might be referred to as an emitter, source, transmitter, or target. For present purposes, we adopt the first of these terms. A single feature generator may produce multiple features, including features pertaining to different modalities, acoustic and electromagnetic waves for example.

A *feature propagator* calculates effect of the environment on the propagation of signature feature statistics. The results of such transmission calculations generally depend on the target/sensor geometry as well as details of the terrain and atmospheric environments. Signal transmission is generally a linear phenomenon, which means that the transmitters operate on multiple features independently and in parallel.

The process of feature generation and propagation applies to emitters of interest as well as interfering background noise (clutter). In fact, the same computational models may be used when appropriate.

While the operations of feature generation and propagation are conveniently conceptualized as distinct from a physical point of view, from the standpoint of software implementation they can be viewed as a single operation, the purpose of which is to produce signal models at the location of the receiver. We term this operation *feature transmission*. Hence, in the terminology of EASEE, transmission combines feature generation and propagation. (In other contexts, the word *transmission* may apply to either signal generation, propagation, or a combination; here, we use it strictly for the combination.)

A *feature sensor* gathers and combines multiple transmitted signals. It can also apply a transfer function to the signal; that is, the levels of the features can be adjusted relative to one another to reflect the response of the sensor in different frequency regions. The feature sensor can also add signal model objects representing the sensor self noise (e.g., noise introduced by the pre-amplifier in a microphone).

A *feature processor* analyzes one or more statistical descriptions of signature features, from which it draws *inferences* (predictions) about the performance of a sensor system. Depending on the context, the feature processor might be referred to as a sensor or receiver. An inference is, essentially, the information desired from a sensor system.

Conceptually, feature sensing and processing may be viewed as a single function, much as feature generation and propagation were earlier viewed as a single function (feature transmission). The combined function of feature sensing and processing could likewise be called *feature reception*.

Besides being drawn directly from signature features, inferences may possibly be drawn from other inferences. For example, the probability of detection of a sensor network might be calculated from probabilities of detection of individual feature processors. We call a processor that draws inferences from other inferences a *fusion processor*.

Software implementation

Java interfaces[3] are used to represent particular capabilities for transmitting signature data features and producing inferences. A feature transmitter, feature receiver, or fusion processor by definition implements one of these interfaces. The scope of the interfaces is illustrated by the dashed lines in Figure 4. The interfaces follow an inheritance tree structure somewhat like (but not completely paralleling) the corresponding trees for signal models and inferences shown in Figure 2 and Figure 3.

A feature transmitter implements an interface describing a method called `transmitFeatures`, which specifies the feature enumerations to be transmitted. For example, the `OpticalSignalTransmission` interface describes the `transmitFeatures` method needed to transmit signature data features belonging to `OpticalSignalFeatures`.

Since feature receivers and fusion processors both produce inferences, they use an overlapping set of interfaces. For example, `DetectionProcessing` governs the implementation of any object that produces detection inferences, whether these are produced by a feature or fusion processor. It specifies a method called `getProcessorPd`, which produces detection inferences. But, not all inferences can sensibly be produced by both feature and fusion processors. For example, the `LocationProcessing` interface would likely be implemented only by a fusion processor, such as an algorithm that fuses target bearings from multiple feature processors.

The interfaces are generally configured to produce Java `ArrayList` types. An `ArrayList` is a collection of objects of a certain class or its subclasses. For example, the `transmitFeatures` method in the `OpticalSignalTransmission` interface produces an `ArrayList` of `AbstractSignalModel` objects. Each of these objects corresponds to the output of a distinct signal generator/propagator combination. This signal-model collection can then be combined with collections produced by other transmitters, and sent to an implementation of `DetectionProcessing`, which determines the probability of detection.

[3] A Java interface is a class consisting entirely of abstract methods. Subclasses implement (provide non-abstract) versions of the methods. The interface programming technique is useful for standardizing the names and input arguments used by methods.

The interfaces encapsulate various objects described in the previous section, as shown in Figure 4. There are abstract base classes called `FeatureGenerator`, `FeaturePropagator`, and `FeatureSensor`. These are implemented by non-abstract classes such as `SeismicFeatureGenerator` and `AcousticFeatureSensor`. The latter is extended by the `Microphone` and `HumanListener` classes. Similarly, `OpticalFeatureSensor` is extended by `GenericOpticalSensor` and `HumanViewer`, whereas `SeismicFeatureSensor` is extended by `Geophone`. The `Microphone` and `Geophone` classes realistically capture the sensor properties of typical microphones and geophones.

Feature generators and feature propagators delegate their task to selectable generation and propagation *models*. These models range from very simple to quite complex. The purpose of having a series of selectable models is to allow users to choose between relatively fast, but low fidelity models and relatively slow, but high fidelity models. All generators have available a binary generation model, which simply produces a value of 1 when a feature is selected, and 0 when it is not. The acoustic and seismic generators have models that read measured signatures from a library of data files. Propagators share models for replication (copying of signal models), cylindrical wave spreading (two-dimensional propagation), spherical wave spreading (three-dimensional propagation), and line-of-sight propagation. The replication method is used to copy noise backgrounds to multiple spatial locations. The acoustic feature propagator also has an impedance-plane (acoustically absorbing ground surface) model (Attenborough et al. 1980), and a Crank-Nicholson parabolic equation model (West et al. 1992). More generation and propagation models will be added to EASEE in the future.

The abstract base class for both feature and fusion processors is `InferenceProcessor`. `DetectionFeatureProcessor` and `BearingFeatureProcessor` are examples of feature processors that extend `InferenceProcessor`, whereas `DetectionFusionProcessor` and `LocationFusionProcessor` exemplify fusion processors that extend `InferenceProcessor`.

`DetectionFeatureProcessor` is extended by a number of different classes, each of which provides a different algorithm for calculating probabilities of detection and false alarm. An important example of a

`DetectionFeatureProcessor` is `DetectionNeymanPearson`, which implements the Neyman-Pearson (constant false-alarm rate) criterion (Burdic 1984). Absolute threshold detection, relative threshold detection, error minimization, and Bayes risk minimization have also been coded.

The techniques described in this section, involving interfaces and object collections, facilitate the creation of a very general software design. The main idea is that the various stages in the information flow shown in Figure 4 *are mutually independent when each step is performed at the highest level of abstraction in the signal models and inferences*. This makes code for the various components highly versatile and reusable. For example, when a new signal model class is written, existing feature processing algorithms operating on the abstract signal model class will accept this new model and continue to function without modification. Conversely, when a new processing algorithm is written, it will automatically operate on all existing and future signal models.

6 Platforms: Sensor and Emitter Representation

A *platform* represents an object that transmits and/or receives signals and data. It incorporates feature transmitters, feature receivers, and fusion processors. Some specific examples of platforms that we might be interested in implementing include tactical ground sensor systems such as the Future Combat Systems (FCS), Tactical Unattended Ground Sensor (T-UGS), intruder detection systems such as the Improved Remote Battlefield Sensor System (I-REMBASS), gateway sensor nodes for collecting and fusing data from other sensors, elevated platforms with imagery packages including unmanned aerial vehicles (UAVs), and ground vehicles such as a HMMWV (High Mobility Multipurpose Wheeled Vehicle) or a pick-up truck. Note that platforms may also represent signal transmitters in a more abstract sense, which is rather different from the conventional use of the word *platform* in a military context. For example, a platform could represent the acoustic background noise produced by a roadway, or the infrared image of a tree or terrain behind the target of interest.

Configuration of a platform

Some illustrative configurations of feature transmitters and processors for representing platforms are shown in Figure 5 and Figure 6. The wheeled ground vehicle illustrates a simple platform with signal emitters only. It transmits acoustic features (from the wheels and engine), seismic features (from the wheels), and IR features (from the external body). The UAV illustrates a hybrid emitter/sensor platform. Its engine produces acoustic features that may be heard on the ground; it also has optical and IR cameras. The intrusion detection sensor illustrates a multi-sensor platform that collects acoustic, seismic, and IR data; detection processing is applied to the sensor data individually and then at the platform level. The ground sensor gateway/fusion node illustrates fusion of detection and bearing inferences from other sensor platforms, which are fused and made available for higher-level interpretation.

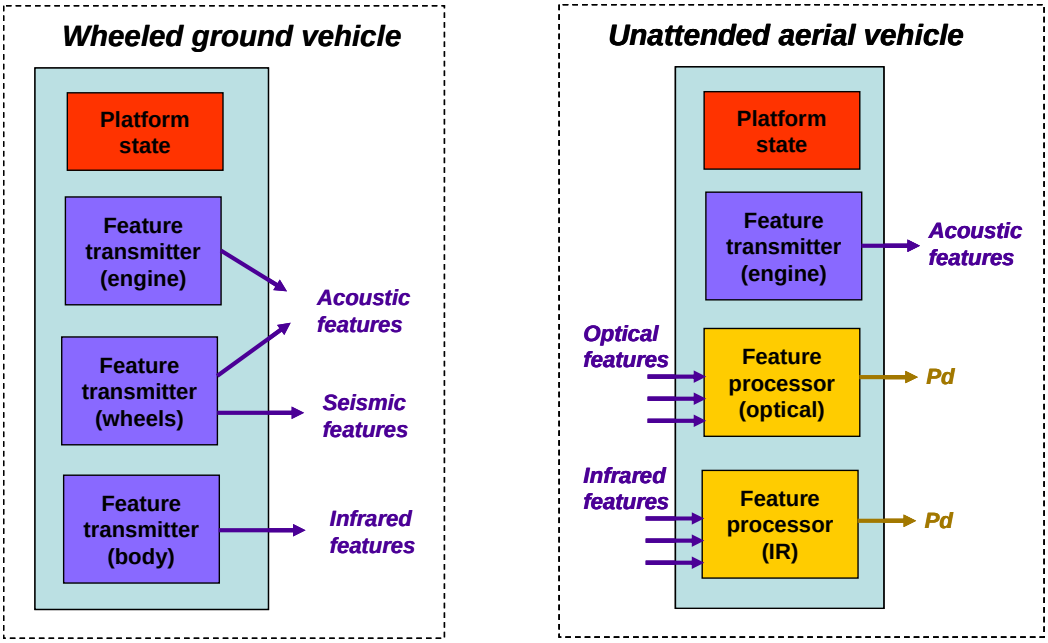


Figure 5. Configurations of feature transmitters and processors to represent a wheeled ground vehicle (left) and an unattended aerial vehicle (right).

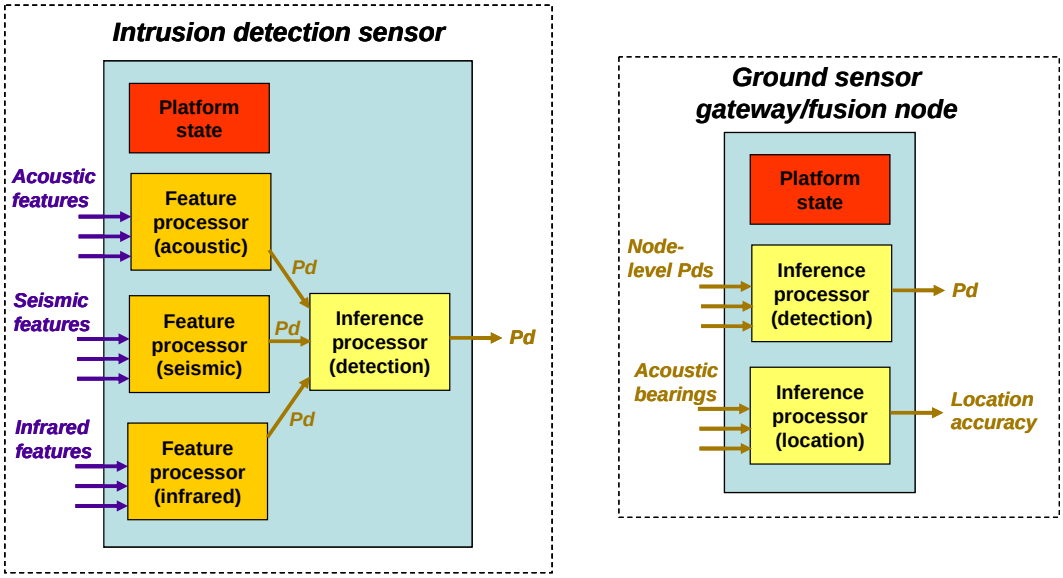


Figure 6. Configurations of feature and fusion processors to represent an intrusion detection sensor (left) and a ground-sensor gateway/fusion node (right).

Software implementation

All EASEE platforms have a *state* that describes the position, velocity, and orientation of the platform as a function of time. Values for these quantities at multiple points in time may be specified, as will be described later in this section. Other state information may also be included, as desired. For example, it might be desirable to include information on the engine

transmission settings of a ground vehicle. The state information can be utilized by feature generators and transmitters to capture dynamics in a particular detection problem, such as the variation of the thermal signature of terrain with time of day.

Just like individual transmitters, receivers, and processors, the platforms implement the feature transmission and inference interfaces described earlier. The main distinction is that the platform may encapsulate multiple transmitters and processors. In the wheeled ground vehicle example in Figure 5, a platform-level implementation of the `AcousticOctaveBandTransmission` interface would encapsulate two transmitters, one for the acoustic emissions of the engine and one for the wheels. The signature data features for each of these transmitters would then be bundled into a Java `ArrayList` collection.

Consistent with the development pattern for other types of objects in EASEE, a generic abstract platform class is at the base of the inheritance tree. This generic platform contains no transmitters, receivers, or fusion processors. But, it does define a pair of lists that play a key role in the EASEE architecture. One listing specifies the platforms to be queried for signature data features, whereas the other specifies the platforms to be queried for inferences. The former can be conceptualized to indicate those platforms in close enough proximity that their signals are able to propagate through the environment and reach the platform of interest. The latter corresponds to those platforms connected through a communication network.

Implementations for specific platform configurations (such as those shown in Figure 5 and Figure 6) add the feature generators, feature propagators, feature sensors, feature processors, and fusion processors, all as needed to create the desired functionality. The general idea is that these platforms represent a general configuration of interest, as illustrated in these figures. Instances of the platform classes are constructed with specific characteristics. For example, instances of wheeled ground vehicles could be a specific type of HMMWV or a Toyota Tacoma pick-up truck. The signature generators would use essentially the same code, but produce somewhat different signatures.

The EASEE platform design and interfaces are configured to support a *data-pull* (as opposed to *data-push*) style of information flow. In a data-

push architecture, the signal emitters (transmitters) produce the signature data features they are capable of producing and then send the data to the sensors (feature processors). In data-pull architecture, the sensors (feature processors) request signature feature data, and then the signal emitters (transmitters) provide it.

An earlier acoustic/seismic DST, the Sensor Performance Evaluator for Battlefield Environments (SPEBE) program (Wilson et al. 2001), implemented a data-push scheme. The main drawback of such a scheme is that, without some additional exchange of information or loss of fidelity, *all* emitter information must be generated and transmitted, so that the sensor platforms can then select what information they need. For example, an emitter platform may include IR, optical, EM, magnetic, acoustic, and seismic signatures. Lacking knowledge of the characteristics of the sensor field, all of this information would have to be pushed. If the sensor field actually consisted only of, say, seismic sensors, this would waste computational resources.

In data-pull architecture, however, the signal processors request and receive only the needed information. Hence we have chosen to implement this approach for EASEE. This approach is also consistent with modern server-client architectures. Overall, the process begins with the user interface application sending a request for a prediction (inference) to the platforms. The platforms then determine which signature data features and inferences will be needed to fill this request, and request the features from other platforms based on the lists mentioned earlier. Eventually, the desired inferences are calculated and sent to the user interface application.

Platform positioning and problem viability

A viable signal transmission and detection problem must contain at least the following three elements: (1) a transmitter of interest (which provides the *signal*), (2) an interfering transmitter (which provides the *noise*), and (3) a receiver (which senses the signal and noise). Many real-world problems actually involve multiple transmitters and receivers. In this section, we consider how many and what combinations of transmitters and receivers should be accommodated in a software design, and the constraints on their allowable positions. This is important for designing an appropriate degree of flexibility into platform representations and employing them.

Let us first consider a straightforward problem in which we have a single transmitter of interest and want to determine at what locations in space it can be detected. (Positioning of the interfering noise transmitter will be discussed momentarily.) This is sometimes called the *footprint of the transmitter*. This situation can be implemented systematically by placing the transmitter at a single fixed position in space, and varying the receiver position across a spatial grid. The density and geometry of the grid are determined by user preferences and computational considerations – too large a grid will result in a too long a calculation. The grid may consist of a regular Cartesian grid or a polygonal terrain representation.

Also of interest is the *reciprocal* of the preceding problem, in which there is a single receiver and we wish to know transmitter locations where detection will occur. This is sometimes called the *footprint of the receiver*. We can implement this problem by placing the receiver at a single fixed position and then varying the transmitter position across a spatial grid.

We refer here to a receiver or transmitter position that is varied across a spatial grid as *variable*. Calculation results corresponding to the position of the variable receiver or transmitter will typically be overlaid on the terrain display as a map layer.

Does it make sense to define a problem in which both the receiver and transmitter positions are variable? The answer is both "yes" and "no." If the positions are varied independently, and there are N locations for each, then there would be N^2 problems to solve. Such a computation could be time-consuming and confusing to display. On the other hand, there may be situations where it is reasonable to vary the positions of the receiver and transmitter in a manner that their relative positions and orientations are fixed. Solution of such a problem involves only N locations. In particular, they could both be at the same grid position. This would make sense if, say, there is a microphone positioned on a vehicle that is emitting acoustic noise, in which case the transmitter and receiver move together.

More complicated problems might involve multiple fixed transmitters and/or multiple fixed receivers. For example, one might be interested in determining the locations where a transmitter could be detected by a network of receivers. In this case, we have multiple fixed receivers but a variable transmitter position. Similarly, we could have multiple fixed transmit-

ters contributing simultaneously to the signals received by a single receiver for which the position is variable.

Another twist to the problem is the existence of interfering transmitters. The interfering transmitters are often referred to as noise or nuisance sources. There could be multiple interfering transmitters. It is also possible to consider an interfering transmitter for which the position is variable. Typically, we would sum the contributions from the interfering transmitters together, and separately sum the contributions from the transmitters of interest; these quantities then represent the signal and noise in the detection problem.

Finally, we might think of a transmitter as being distributed in space. For example, the noise produced by traffic is distributed along a roadway, and the noise produced by wind blowing through a forest might be regarded as distributed in volume. In a numerical implementation, the distributed transmitter could be represented at many discrete locations on a computational grid, and its strength adjusted in accordance with the resolution of the grid. While in some situations such a representation might be useful, typically it will result in a large number of transmitters, the signals of which must be propagated (transmitted) through the environment, and this would likely be a very computationally intensive process. A practical resolution is to represent the actual field produced by the distributed transmitters directly. This is precisely what a receiver would observe and is often measured directly as the "background noise" for a particular problem.

All of the situations described above (except for the N^2 problem) can be addressed in the following way. We solve N problems involving one or more transmitters of interest, one or more interfering transmitters, and one or more receivers in parallel. If desired, each transmitter/receiver can be at a fixed position in space for each of these N problems, or it can vary among locations on a spatial grid.

This observation motivates the structure of the platform state description in EASEE. As mentioned earlier, the state may describe multiple platform locations at each point in time. Specifically, all platforms in a given problem are allowed to have N locations at a given time. It is left to the user application interface driving the EASEE calculation to set up the locations in a manner consistent with the problem posed by the user.

7 Environmental Representation

We use the term *scenario* for the environmental (atmospheric and terrain) information needed by a calculation. In general, the scenario may impact all stages of a sensor performance calculation. For example, an IR signature feature generator is dependent upon factors that include the solar angle, cloud cover, and wind. The signal propagation process is strongly affected by the environmental conditions, particularly for acoustics and seismics. The performance of a seismic sensor is affected by its coupling to the ground, which depends on the ground properties.

One of the main challenges in representing the environment is to accommodate multiple types of data specifications. The weather may be described by data from a numerical forecast model or by selecting from a library of typical weather conditions. Terrain elevation data comes in varying resolutions (e.g., varying levels of digital terrain elevation data [DTED] and grid structures [Cartesian, polygonal, etc.]).

In EASEE, s*cenario translators* play the important role of converting the scenario information into parameters needed by platforms and their feature generators, propagators, and processors. The translators implement Step 2 of the process described in Section 2. Generally speaking, a different translator is needed for each combination of scenario specification and feature generator, propagator, and processor. In practice, coding of scenario translators is a very important and challenging part of the predictive process. It often involves physically based assimilation methods designed to make optimal use of available information and to supply reasonable values for unavailable parameters.

Two environmental scenario classes have been implemented in EASEE. Both are extensions of the abstract parent class `EnvironScenario`, which includes only a time stamp for the environmental data and a digital elevation map describing the terrain elevations.

One of the environmental scenario classes, `EnvironScenarioHomo`, is intended to be the simplest possible description of the environment. It includes objects for the following: (1) the time of the atmospheric/terrain observation, (2) the atmosphere as humid air (a mixture of dry air and wa-

ter vapor, each of which are assumed to be ideal gases), (3) the subsurface as an isotropic, linear, lossless solid, and (4) the local height of the ground above sea level. Some additional information on these object representations is provided in the Appendix.

The other environmental scenario class, `EnvironScenarioVert`, supports specifications of height-dependent atmospheric properties, depth-dependent subsurface properties, and a digital elevation map (DEM). It includes objects for the following:

- the time of the atmospheric/terrain observation
- the vertical profiles of atmospheric wind, temperature, pressure, and humidity
- the low-, mid-, and high-altitude cloud fractions
- an atmospheric surface-layer parameterization for heat, momentum, and humidity exchange with the ground surface (which can be used to describe turbulence)
- a parameterization of the aerodynamic, acoustic, and optical properties of the ground
- a geographic grid representing the DEM
- the vertical profiles of subsurface density, compressional wave speed and attenuation, and shear wave speed and attenuation

Availability of scenario translation for a particular environmental representation is declared at the transmitter, processor, or platform level through implementation of a corresponding interface. `EnvironHomoTranslation` is for translation of homogeneous environmental descriptions, `EnvironVertProfTranslation` is for translations environmental descriptions with vertical profiles of atmospheric and subsurface variables. When a platform supports a specified interface, all signal transmitters and processors on the platform must also support that interface.

8 Signal Transmission Grids

The EASEE architecture has been designed to accommodate signal propagation models possessing varying degrees of fidelity. To this end, signal propagation is described by *transmission grids* possessing various spatial symmetry properties. The grid describes how the signal depends with distance and direction from the source, and also how it depends on the source position, if this is indeed the case. The parameters stored on the transmission grid are signal power (or another conservative quantity such as mass), and the azimuthal and elevation angles of the propagation direction. The appropriateness of a given grid will depend on the signal modality (e.g., acoustic, seismic, radio frequency, optical, etc.) as well as the environmental model. Grids of lower complexity can generally be promoted to grids of higher complexity.

Transmission grids are available in two general varieties: *structured* and *unstructured*. Structured grids are appropriate when the source and/or receiver positions occur in a regular, geometric pattern. Unstructured grids are appropriate when the positions are irregular. In EASEE, structured grids are also intended for calculations performed in free space or above a flat ground, because calculations performed on these grids do not incorporate information on the digital elevation map (terrain elevations). Calculations performed on unstructured grids do incorporate the DEM.

The simplest type of structured grid is for propagation problems in which the signal strength depends only on the distance from the source to the receiver; that is, the propagation is independent of the source position and the direction to the receiver. Since the propagation thus depends on only one variable (distance), the calculations can be stored in a one-dimensional array. We call this format a *homogeneous, isotropic* grid. Such a format might be appropriate if the environment is horizontally stratified, if the signal propagation is unaffected by the ground, and if there are no significant horizontal variations in the atmosphere.

The next considered degree of complexity is for problems in which the propagation depends on the height of the source and the height of the receiver, as well as the horizontal distance (called the *range*) between them. This situation applies, for example, to acoustic or electromagnetic propa-

gation in an atmosphere with horizontally stratified density or thermal structure, and to seismic propagation when the ground has horizontally stratified density, wave speeds, and bulk moduli. This type of grid is termed *vertically inhomogeneous* and *horizontally isotropic* and is 3D in its storage requirements.

In some situations, such as sound propagation in the presence of wind, it becomes important to account for the horizontal directionality of the propagation. Hence the next degree of complexity is the *vertically inhomogeneous* and *horizontally anisotropic* grid, which involves 4D storage. Here, the propagation depends on the source height, receiver height, range, and azimuthal direction of the receiver relative to the source.

All of the grids discussed to this point have assumed that the propagation does not depend on the absolute position of the source. Such an assumption breaks down when there is substantial horizontal variability in the environment. In such situations we must resort to the most general form of grid, namely where the signal propagation depends on the 3-D coordinates of both the source and receiver. The storage requirements in general include the x, y, and z Cartesian coordinates of both the source and the receiver, and thus are 6-D. EASEE incorporates one variety of structured 6-D grid, namely a Cartesian (rectangular) grid mesh. This format can be applied to situations where each of the coordinate axes of the sources and receivers is independent.

As mentioned earlier, for the unstructured grids, the source and receiver coordinates need not follow any regular pattern. EASEE incorporates two general formats of unstructured grids: *dual unstructured* and *fully unstructured*. The dual unstructured grid specifies the source and receiver locations independently. A calculation is performed for each combination of source and receiver locations. Hence, if N_s source locations are specified, and N_r receiver locations are specified, the number of grid elements is $N_s\, N_r$.

The fully unstructured grid has a single list that specifies pairs of source and receiver positions; that is, N pairs of source and receiver positions are specified. The transmission is calculated for each of the N pairs of source and receiver positions.

The transmission grids may be constructed either by specifying only the signal power, or by specifying the signal power as well as the propagation directions. If the directions are not specified, propagation is assumed to be line-of-sight. That is, the direction of the receiver relative to the source is calculated, and used to infer the propagation direction.

Propagation calculations in EASEE are actually explicitly performed on unstructured grids. If a single source or receiver position enters into the calculation, a dual unstructured grid method is called. If multiple (but equal in number) source and receiver positions are specified, as discussed in Sec. 6, a fully unstructured grid is used. A new signal model is then constructed at each grid location. The reason for emphasizing the unstructured grid is that the source and receiver locations may be arbitrary, or they may be distributed across a geographic grid, such as a latitude/longitude coordinate system, that does not lead to equal spacing.

Still, the structured grids play an important, underlying role in EASEE. Propagation models often naturally produce structured grids. For example, the previously mentioned Crank-Nicholson parabolic equation naturally produces vertically inhomogeneous and horizontally anisotropic (4D) grid. It is thus convenient to perform and store the calculation on this grid, rather than a full 6D grid. The calculation results can then be interpolated onto an unstructured grid as needed.

9 Conclusions

The purpose of this report has been to describe a flexible, object-oriented software design for predicting signal transmission and sensing on the battlefield. The Java-language software based on this design is called Environmental Awareness for Sensor and Emitter Employment, or EASEE.

The central elements to the EASEE software design are statistical models for *signature data features* and *inferences*. EASEE provides the "glue" for tying together models for signature generation, propagation, sensing, and processing, in a manner that decouples these models from specific environmental descriptions.

At the time of this writing, an initial version of a MATLAB user interface for EASEE has been completed. We have begun but have not yet completed implementation of a Java OpenMap implementation. Our intention is to make EASEE general enough so that it can be implemented within many other software architectures, and thus enable advanced signal propagation and processing calculations in a great variety of modeling and simulation efforts supporting the warfighter and homeland security.

References

Attenborough, K., S. I. Hayek, and J. M. Lawther. 1980. Propagation of sound over a porous half-space. *J. Acoust. Soc. Am*. 68: 1493–1501.

Baker, M. B, M. Eylander, and H. Harrison. 1983. The statistics of chemical trace concentrations in the steady state. *Atmos. Environ*. 18: 969-975.

Burdic, W. S. 1984. *Underwater acoustic system analysis*. Englewood Cliffs NJ: Prentice-Hall. pp. 244–296.

Frankenstein, S., and G. G. Koenig. 2004. *Fast all-season soil strength (FASST).* ERDC/CRREL SR-04-1. Hanover NH.: U. S. Army Corps of Engineers, Cold Regions Research and Engineering Laboratory.

Hieb, M. R., S. Mackay, M. W. Powers, H. Yu, M. Kleiner, and J. M. Pullen. 2007. *Geospatial challenges in a net centric environment: Actionable information technology, design, and implementation*, report 657816. In proceedings of SPIE Defense and Security Symposium, Defense Transformation and Net-Centric Systems, edited by R. Suresh.

Klein, L. A. 2004. *Sensor and data fusion*. Bellingham, WA: SPIE Press,.

Limpert, K., W. A. Stahel, and M. Abbt. 2001. Log-normal distributions across the sciences: Keys and clues. *BioScience* 51: 341-352.

Lowe, D. 2005. *Java all-in-one desk reference for dummies*. Hoboken NJ: Wiley. p 860.

Nadarajah, S. 2008. A review of results on sums of random variables. *Acta Appl. Math* 103: 131–140.

West, M., K. Gilbert, and R. A. Sack. 1992. A tutorial on the parabolic equation (PE) model used for long range sound propagation in the atmosphere. *Appl. Acoustics* 37: 31–49.

Wilks, D. S. 2005. *Statistical methods in the atmospheric sciences*. Burlington, MA: Academic Press.

Wilson, D. K., D. H. Marlin, and S. Mackay. 2007. *Acoustic/seismic signal propagation and sensor performance modeling*, report 65620R. In proceedings of SPIE Defense and Security Symposium, Unattended Ground, Sea, and Air Sensor Technologies and Applications IX, edited by E.M. Carapezza.

Wilson, D. K., J. M. Noble, B. H. VanAartsen, and G. L. Szeto. 2001. *Battlefield decision aid for acoustical ground sensors with interface to meteorological data sources* , 208–219. In proceedings of SPIE AeroSense Symposium, Battlespace Digitization and Network-Centric Warfare, edited by R. Suresh.

Wilson, D. K., C. L. Pettit, M. S. Lewis, S. Mackay, and P. M. Seman. 2008. *Probabilistic framework for characterizing uncertainty in the performance of networked battlefield sensors*, report 698104. In proceedings of SPIE Defense and Security Symposium, Defense Transformation and Net-Centric Systems, edited by R. Suresh..

Wilson, D. K., B. M. Sadler, and T. Pham. 2002. *Simulation of detection and beamforming with acoustical ground sensors*, 50–61. In proceedings of SPIE AeroSense Symposium, Unattended Ground Sensor Technologies and Applications IV, edited by E.M. Carapezza.

Wilson, D. K. and J. I. Torrey. 2006. *Object-oriented approach to manipulating acoustic and seismic spectra*. ERDC/CRREL TR-06-20, U. S. Army Corps of Engineers, Cold Regions Research and Engineering Laboratory, Hanover NH. 34 pp.

Appendix: Overview of EASEE Java Packages

This appendix provides an overview of the Java packages comprising EASEE. It describes the various packages and classes at the time of this report's writing. The EASEE software is expected to continue to evolve.

Up-to-date documentation of a more technical nature, describing in detail the various classes and their fields and methods, is distributed with the software in conventional JavaDoc format.

Package mil.army.usace.easee.signalmodels

This package defines the signal models used to describe statistics of the signature data features. Much background on this package has already been provided in Section 3. For illustrative purposes, Figure 7 shows the branch of the inheritance tree leading to `GammaSignalModel`, along with the particular methods defined at each stage.

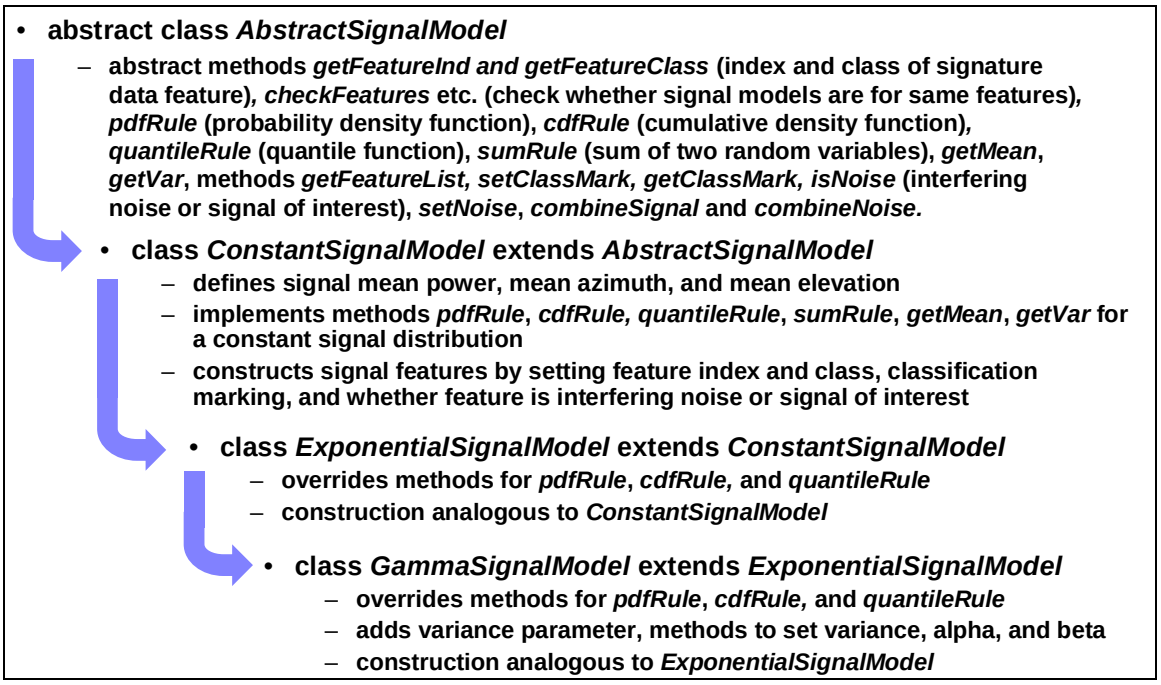


Figure 7. Derivation of the *GammaSignalModel* class, showing its parent classes and inherited methods.

Package mil.army.usace.easee.spectra

This package provides a flexible representation for spectra as a sequence of bands, each given by a frequency-dependent power law. The approach is described by Wilson and Torrey (2006).

At present, the banded power-law representation is used mainly to store acoustic and seismic spectra, and to convert them to signature data features as needed. However, this representation may be used in the future as a basis for building more flexible models for signature data features possessing variable center frequencies and bandwidths.

Package mil.army.usace.easee.infermodels

This package deals with inference modeling. Much of its functionality was described in Section 4. Here, Figure 8 illustrates the inheritance tree for processing features into detection inferences, along with the particular methods defined at each stage.

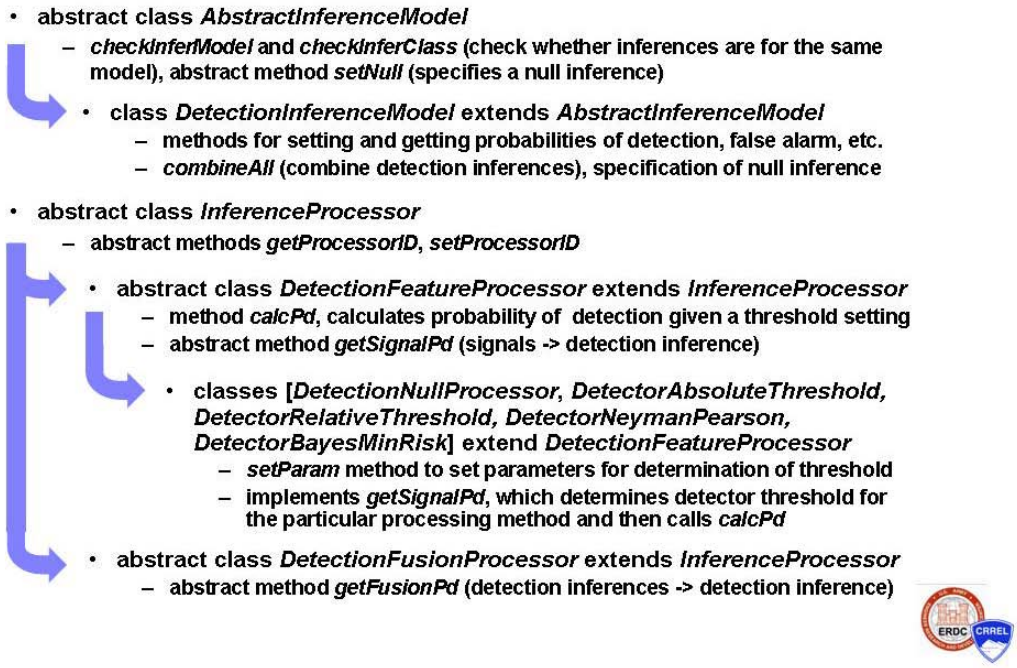


Figure 8. Derivation of the detection feature and fusion processors, showing parent classes and inherited methods.

Package mil.army.usace.easee.transmit

The `transmit` package contains the classes for representing and manipulating signal transmission grids as described in Section 8. It also includes signal propagation models that are common to different kinds of signal types. These include methods for signal replication, cylindrical and spherical wave spreading, and for line-of-sight analysis.

Packages mil.army.usace.easee.acoustic, mil.army.usace.easee.seismic, mil.army.usace.easee.optical, and mil.army.usace.easee.radiofreq

Some background pertinent to these packages, which implement specific signal modalities, was provided in Section 5. These packages consist of five basic components.

1. Java enumeration classes defining the signal features and their properties.
2. A feature generator class (extension of the base `FeatureGenerator` class) specific to the signal modality. This feature generator might call one or more generation models included in the package.
3. A feature propagator class (extension of the base `FeaturePropagator` class) specific to the signal modality. This feature propagator might call one or more propagation models included in the package.
4. A feature sensor class (extension of the base `FeatureSensor` class) specific to the signal modality. One or more extensions to this feature sensor may be provided to implement particular types of sensors.
5. Descriptions of environmental properties specific to the signal modality.

To clarify this last item, we focus on the `acoustic` package as an example. This package defines an abstract class called `AcousticMedium`. This class includes methods for retrieving and calculating the acoustic wave number, attenuation, phase speed, characteristic impedance, complex bulk modulus, and complex bulk density. Several classes extend these methods. Of these, `FluidMedium` is the simplest. It provides a two-parameter representation of a lossless fluid described by the fluid density and bulk modulus. Another extension, `AirMedium`, builds on the `HumidAir` class (from the `EnvironScenario` package) by adding methods to calculate the sound speed and acoustic attenuation in humid air. `RelaxMedium` and `ZwikkerKostenMedium` provide acoustic properties appropriate for propagation in a porous material such as soil or snow.

Package mil.army.usace.easee.platform

Platforms were described in Section 6. As indicated there, the abstract `Platform` class serves as the basis for all other platform designs. A platform includes a specification of its state, which is an object of the `PlatformState` class, to be described shortly. Signal generators and propagators, sensors, and feature and fusion processors, are also instantiated by the platform. Platforms also include links for signature data feature and inference requests, as described in Section 6.

The `PlatformState` class contains an array of times. The initial time in the array specifies the time at which the platform is assumed to come into existence. Subsequent times represent changes in state. For each time, new spatial coordinates, velocities, attitude angles, and state variables are specified. The coordinates are specified as a `GeoGrid3D` object, which is part of the `timeandspace` package. The interpretation of the state variables is specific to a platform.

Package mil.army.usace.easee.timeandspace

This package includes a number of classes for manipulating time and space coordinates.

The `TimeCoord` class is basically a wrapper around a `GregorianCalendar` object, which is a part of the Java API (Application Interface). It simplifies setting and retrieval of dates and times, conversions between Universal Coordinated Time (UTC) and local time, and provides the date and time in a string format that may be used by software that interfaces to EASEE.

The `GeoCoord` class specifies geographic coordinates on the Earth's surface. The coordinates can be set or retrieved in latitude/longitude or Universal Transverse Mercator (UTM) formats. Functionality is also provided to calculate the distance, relative northing, and relative easting between multiple points. `GeoCoord3D` extends `GeoCoord` to include an altitude specification which may be specified as either height above sea level or height above local ground level.

The `GeoGrid` class provides a two-dimensional grid of latitudes and longitudes. The origin and spacing of the grid are set by the user. Similarly,

`GeoGrid3D` is a three-dimensional grid, with latitude, longitude, and altitude.

The `ViewShed` contains static methods for converting between height above ground level (AGL) and above sea level (ASL), for analyzing lines of sight, and for determining viewing angles on varying terrain.

Package mil.army.usace.easee.environscenario

This package provides the environmental representation capability as described in Section 7. It includes some capabilities that may be of general interest for atmospheric and seismic modeling.

The `AtmosVertProf` class defines mean vertical profiles for the wind, temperature, humidity, and pressure in the atmosphere. It includes methods for setting and retrieving the profiles, and for converting between different wind coordinate systems, temperature systems, and humidity representations.

The `HumidAir` class represents the properties of dry air mixed with water vapor. It defines a number of constants for thermodynamic properties of dry air and water vapor, and methods for getting and setting the temperature, pressure, density, and humidity. (Note that, for an ideal gas, only two quantities among the temperature, pressure, and density are independent.) Humidity may be specified as mixing ratio, specific humidity, relative humidity, and other options; the class converts between these representations. Methods are also provided for calculating the viscosity, thermal conductivity, Prandtl number, and saturation vapor pressure. The `HumidAir` class complements `AtmosVertProf`, since it allows quantities to be calculated, such as density, that are not explicitly modeled as vertical profiles in `AtmosVertProf`.

The `AtmosSurfLayer` class represents a turbulent, constant-flux atmospheric surface layer (ASL). (Typically, the surface layer occupies the lowermost 30–300 m of the atmosphere.) The class defines the parameters necessary to represent a turbulent ASL, including the friction velocity, sensible heat flux, latent heat flux, roughness length, and wind direction. Constructors are generally used to set these fields. Then, the near-ground wind, temperature, and humidity profiles can be retrieved.

The `CloudFrac` class is a very simple one, as it represents the fraction of the sky that is blocked by clouds at low, middle, and high altitudes.

The `GroundSurface` class represents the hydraulic, optical, and acoustical properties of the ground surface. (Note that aerodynamic properties of the ground are defined in the `AtmosSurfLayer` class.) Among these are the permeability, static flow resistivity, volume porosity, emissivity, and albedo. Typical values for these quantities are provided for common ground surfaces such as grass, sand, and snow.

The counterpart of `AtmosVertProf`, but for subsurface properties, is the `SeismicVertProf` class. It defines vertical profiles for the compressional wave speed, shear-wave speed, compressional wave attenuation quality factor, shear-wave attenuation quality factor, and density in the ground.

The `SolidIsoLinear` complements `SeismicVertProf` much as `HumidAir` complements `AtmosVertProf`. `SolidIsoLinear` includes typical values for the density, compressional wave speed, and shear-wave speed of a number of typical ground types such as water, concrete, sand, shale, basalt, etc. Objects may be created by specifying density, compressional wave speed, and shear-wave speed or by density, bulk modulus, and shear modulus. Methods are provided for retrieving the wave speeds, moduli, Lamé constants, Poisson's ratio, etc.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
30-12-2009	Final	

4. TITLE AND SUBTITLE
Object-Oriented Software Model for Battlefield Signal Transmission and Sensing

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
D. Keith Wilson, Richard Bates, and Kenneth K. Yamamoto

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Research and Development Center (ERDC)
Cold Regions Research and Engineering Laboratory (CRREL)
72 Lyme Road
Hanover, NH 03755-1290

8. PERFORMING ORGANIZATION REPORT NUMBER
ERDC/CRREL TR-09-17

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Research and Development Center (ERDC)
Geospatial Intelligence Program, Exploitation in Man-made Environments
72 Lyme Road
Hanover, NH 03755-1290

10. SPONSOR/MONITOR'S ACRONYM(S)
GEMENI

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Realistic modeling and simulation of battlefield signal transmission and sensing requires accounting for many complicated environmental and mission-related factors. These factors present complex challenges because of their diversity. Yet, successful modeling and simulation can enable effective mission planning and can support a variety of other military objectives. This report describes the development of a very flexible, object-oriented software design for predicting signal transmission and sensing on the battlefield. This Java-language software is called Environmental Awareness for Sensor and Emitter Employment (EASEE). It is intended for application to a wide range of sensing modalities, and for incorporation into military command and control (C2) systems, decision support tools (DSTs), and force-on-force simulations. An initial version of a user interface for EASEE has been completed and a Java OpenMap implementation has begun. The goal of this work is to make EASEE general enough to be implemented within many software architectures, which will enable advanced signal propagation and processing calculations in a number of modeling and simulation efforts in support of the warfighter and homeland security.

15. SUBJECT TERMS
Software, modeling and simulation, battlefield sensors, EASEE, decision support tools (DSTs), object-oriented programming (OOP), military training

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER (include area code)
Unclassified	Unclassified	Unclassified	UU	45	